Deep Learning with CUDA
Deeper into Deep Learning

Tomasz Szumlak

WFiIS AGH

13-20/03/2024, Kraków

Parameters initialisation

- On the surface, the initialisation may seem to be a trivial aspect of the overall training procedure
- However, a good strategy in choosing the initial values for the weights may lead to the success or failure of our model
  - Incredible story of AlexNet crushing the other models in 2012

- Let's use the intuition and experience we gained when experimenting with TensorFlow playground
- What does it mean that the **weights are large/small**?
- Is it optimal to **set the initial weights and biases to zero**?

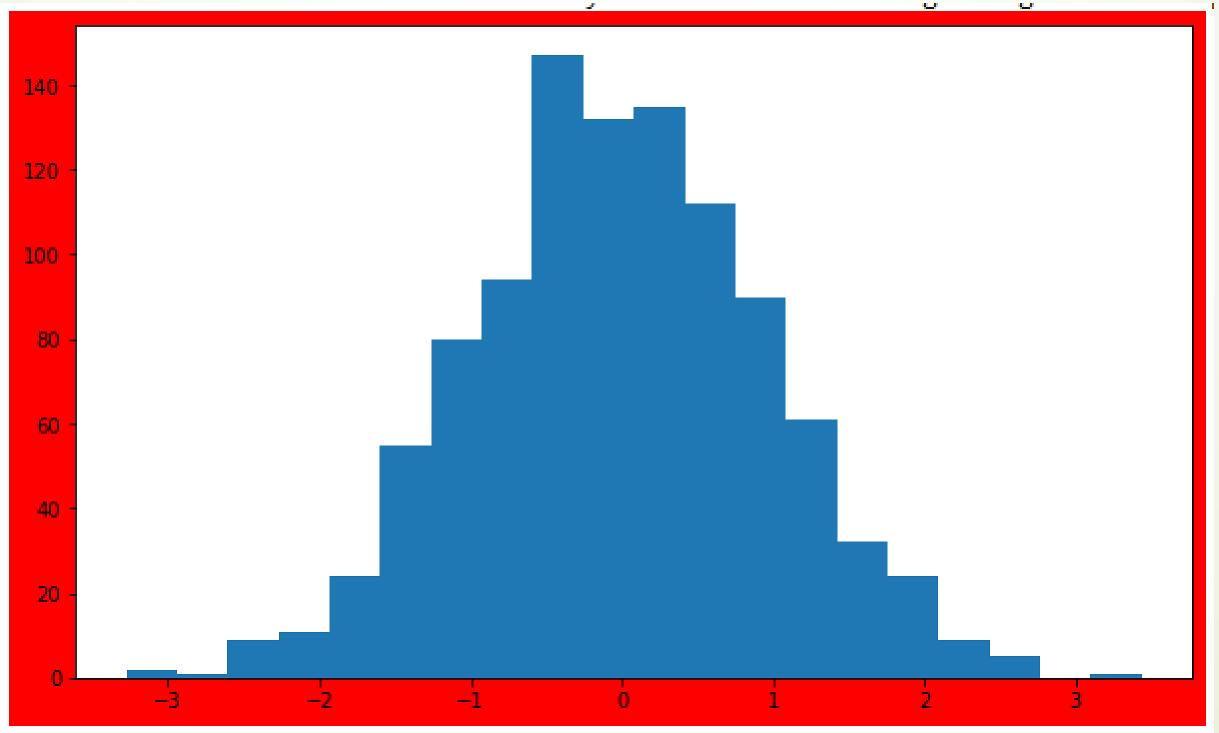
Parameters initialisation

- Large initial values would suggest to the model that there is **already some knowledge accumulated** – that is bad
- **All weights set to zero** may have potentially very strong adverse effect on learning

- Seems that the best strategy would be to use random initialisation and set all the biases to 0, however we are in for some surprises…

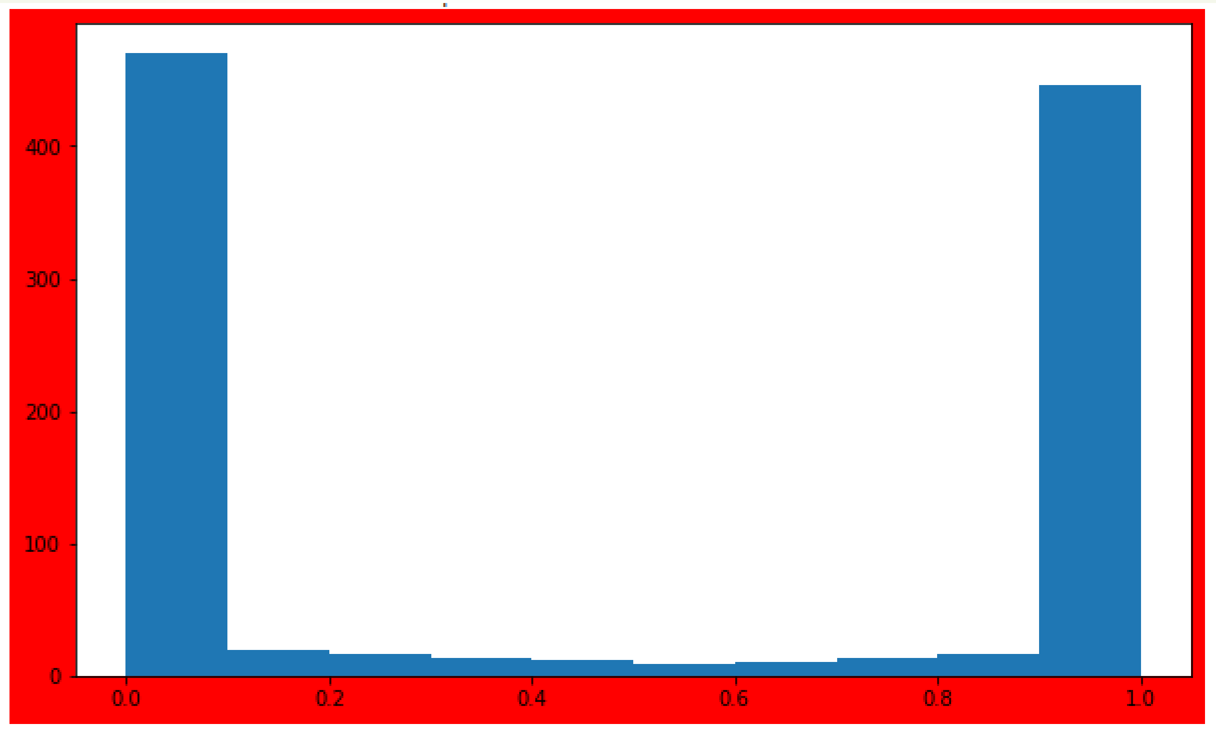
Naive – normal initialisation

- Can perform random initialisation using Keras RandomNormal generator



Naive – normal initialisation

- ❑ Lets propagate simulated random signal forward

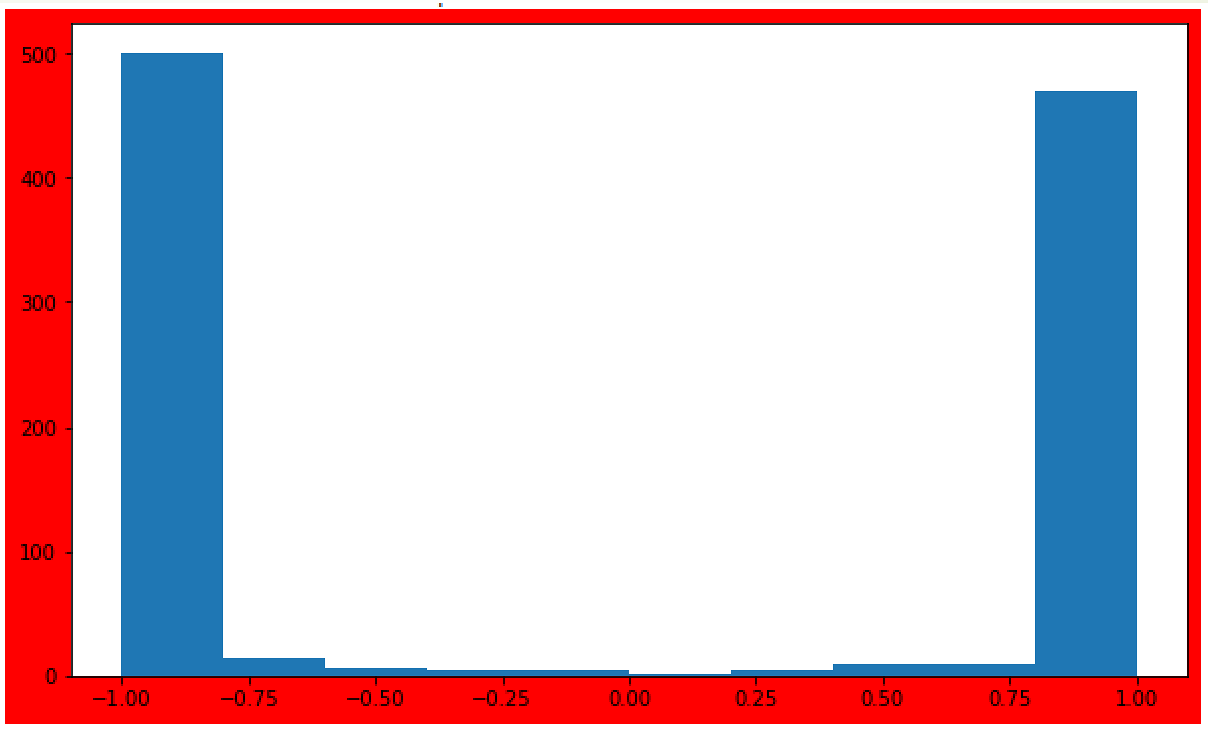


Naive – normal initialisation

- This may at first feel counterintuitive – normally distributed parameters result in obtaining quite particular neuron responses
- Most of them are clustered close to „0" or „1", this shows strong tendency to „saturate" neurons
- We can check, that such behaviour is consistent for other types of neurons we discussed (tanh and relu)

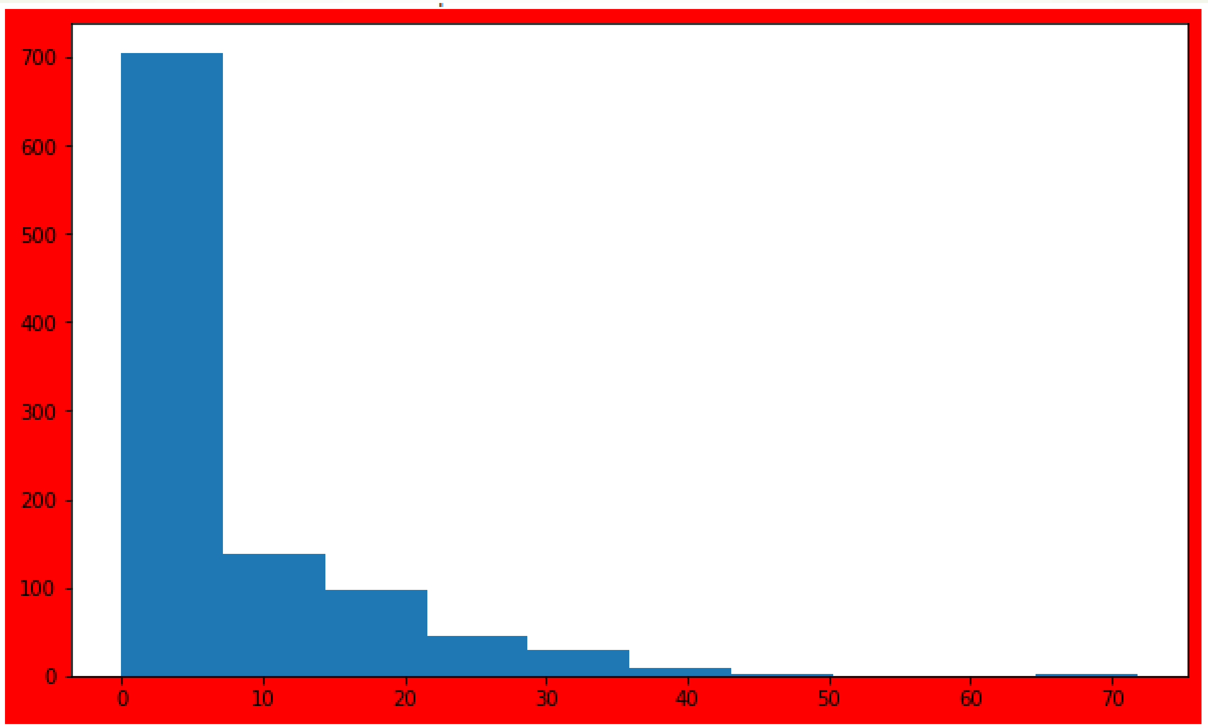
Naive – normal initialisation

- Similar response for **tanh neuron**… (note the range!)



Naive – normal initialisation

- And now for the **relu neuron**… (note the range!)



Naive – normal initialisation

- When sampling from the normal distribution the effects related with forward signal propagation are not good
- Sigmoid and tanh neurons have clear **tendency for strong saturation**
- ReLU neuron shows strongly asymmetric response

- We could interpret these observations as z variable being extremal – either very large or very small (try to think about the respective shapes of the response curves)
- It seems, **that our network has very strong opinion on the data from the beginning**
  - So, the network must do some un-learning in order to learn…

Xavier Glorot to the rescue!

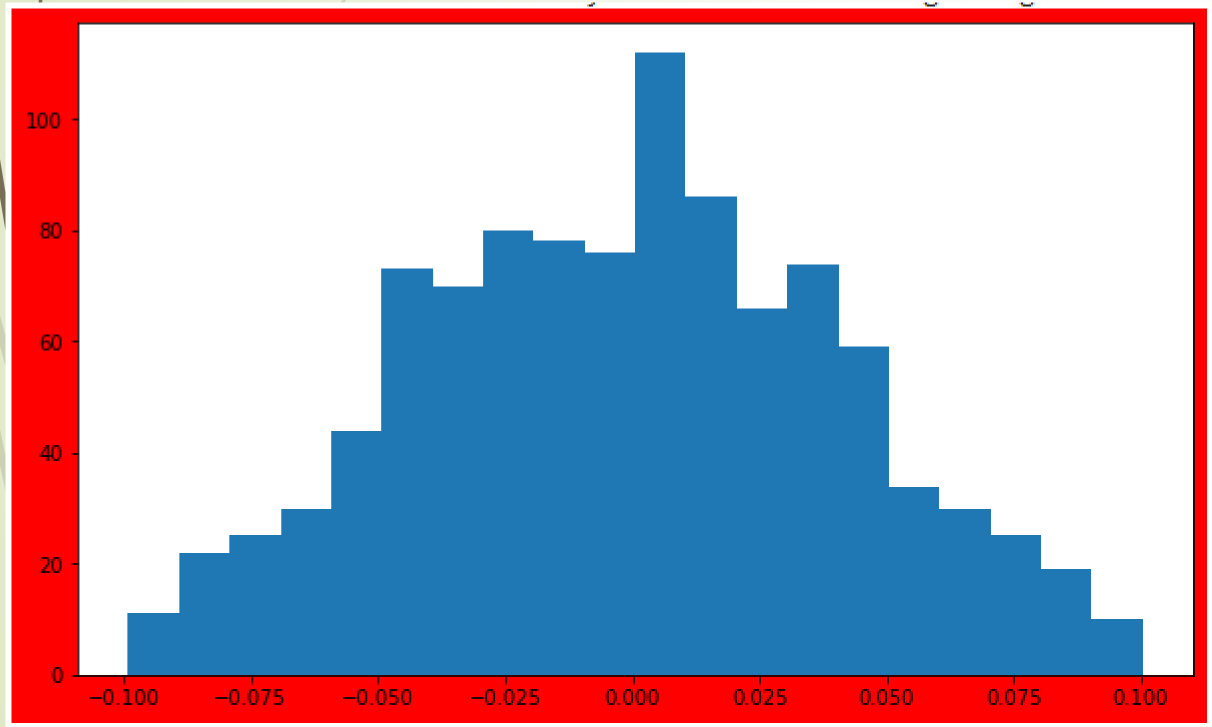
- Optimal distribution function to sample the initial weights can be worked out using the assumptions that the network should work with signal gain close to one
- In other words: signal for both forward and backward propagation should flow with constant variance

- The sketch of the proof providing the optimal initialisation function can be found here: https://towardsdatascience.com/xavier-glorot-initialization-in-neural-networks-math-proof-4682bf5c6ec3

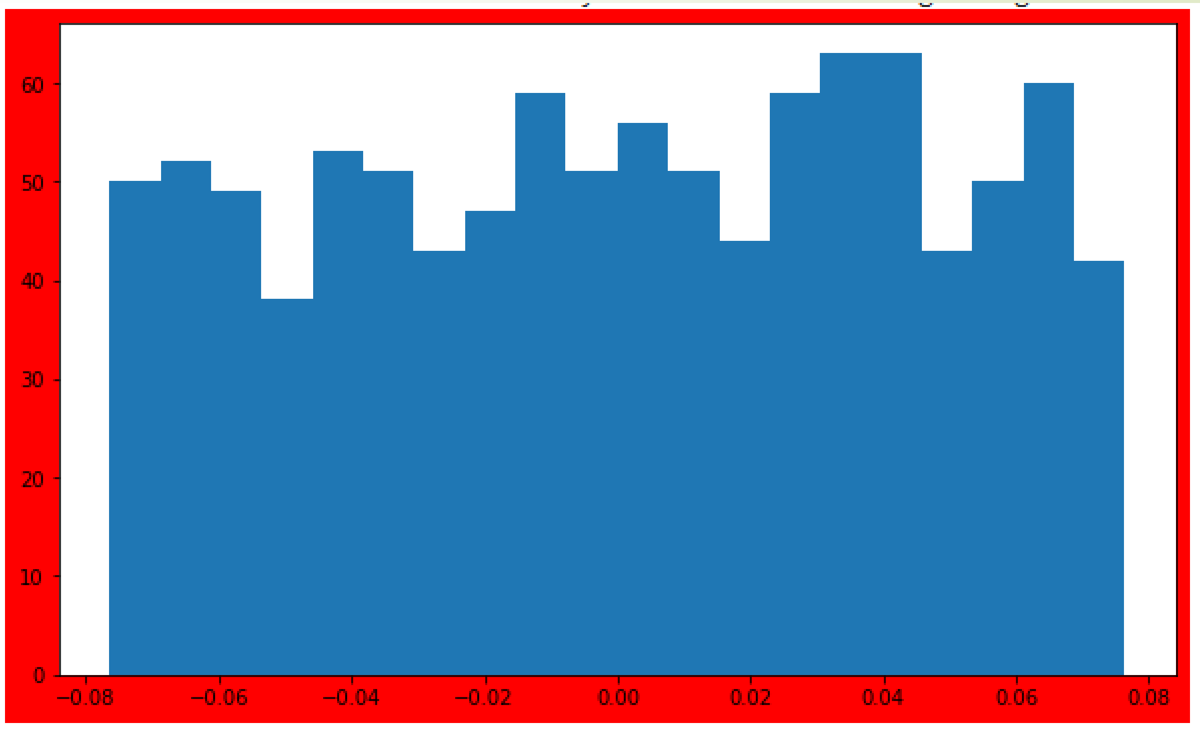
- In Keras we have two types of glorot functions

Glorot function flavours

glorot_normal

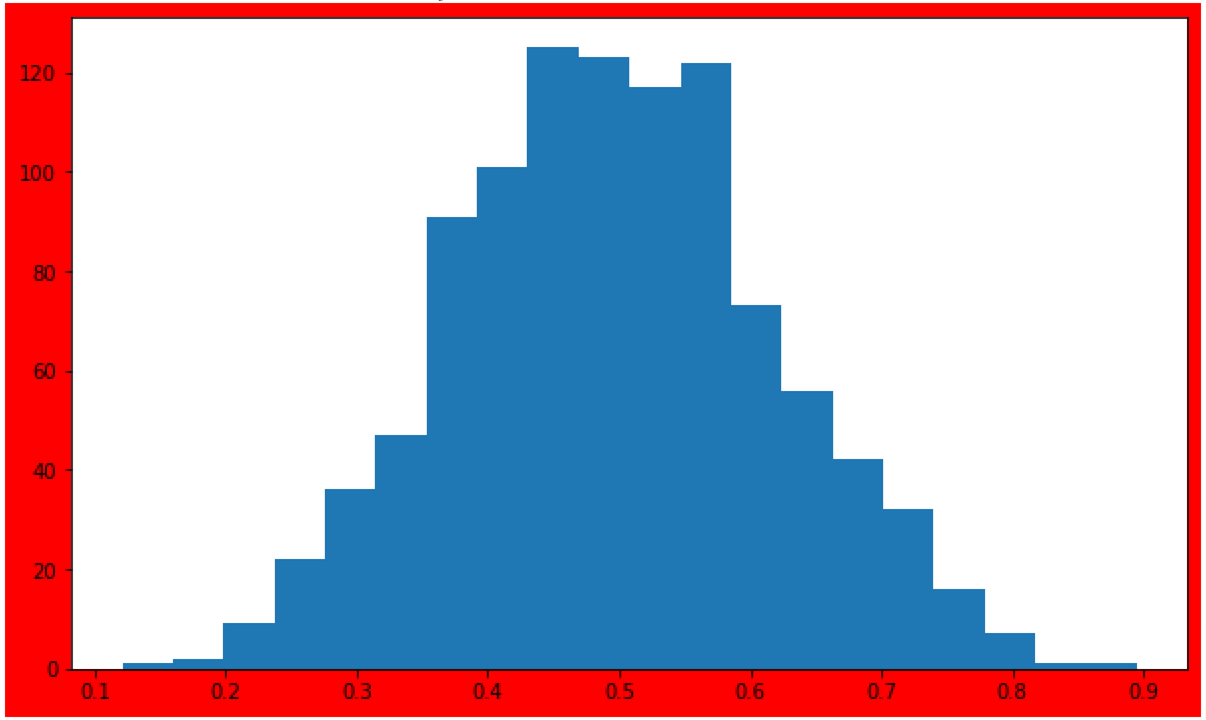


glorot_uniform

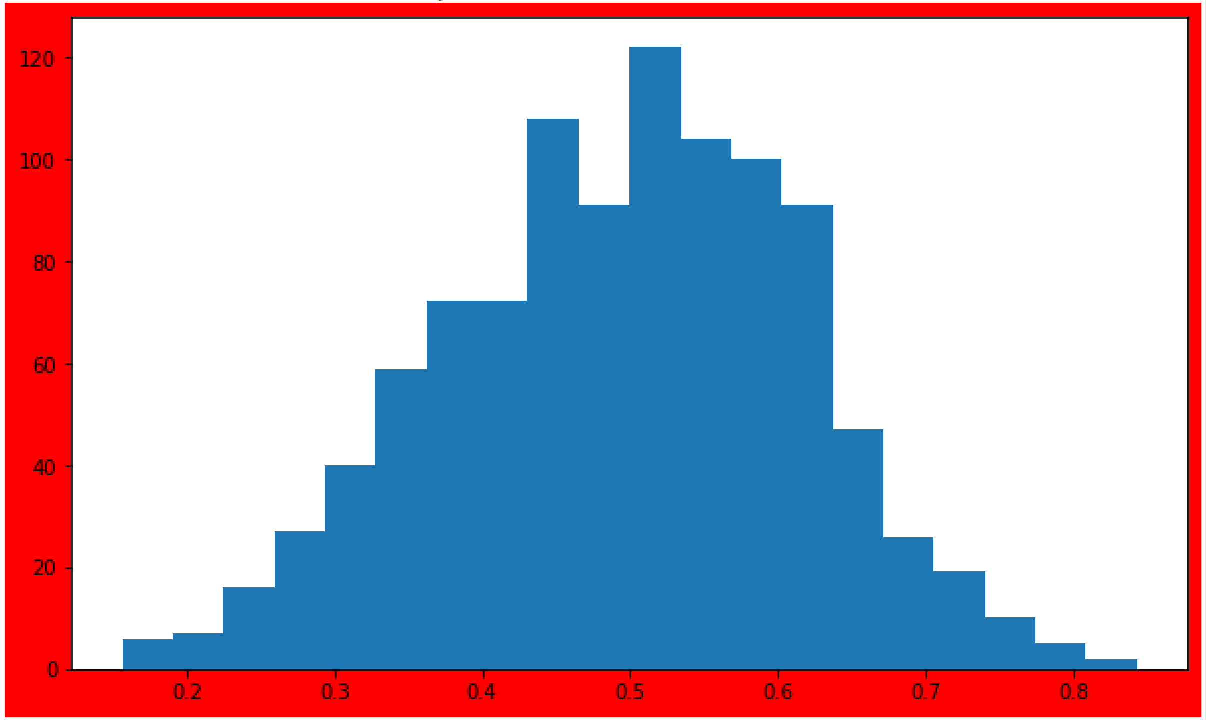


Glorot + sigmoid

glorot_uniform

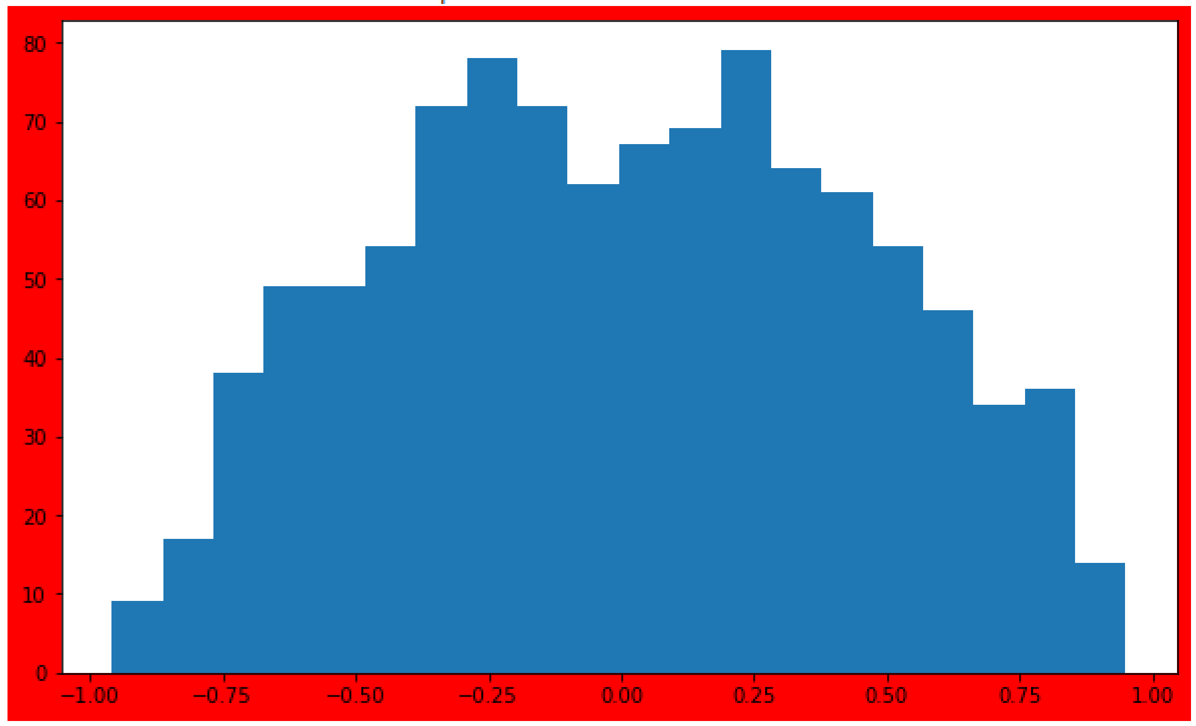


glorot_normal

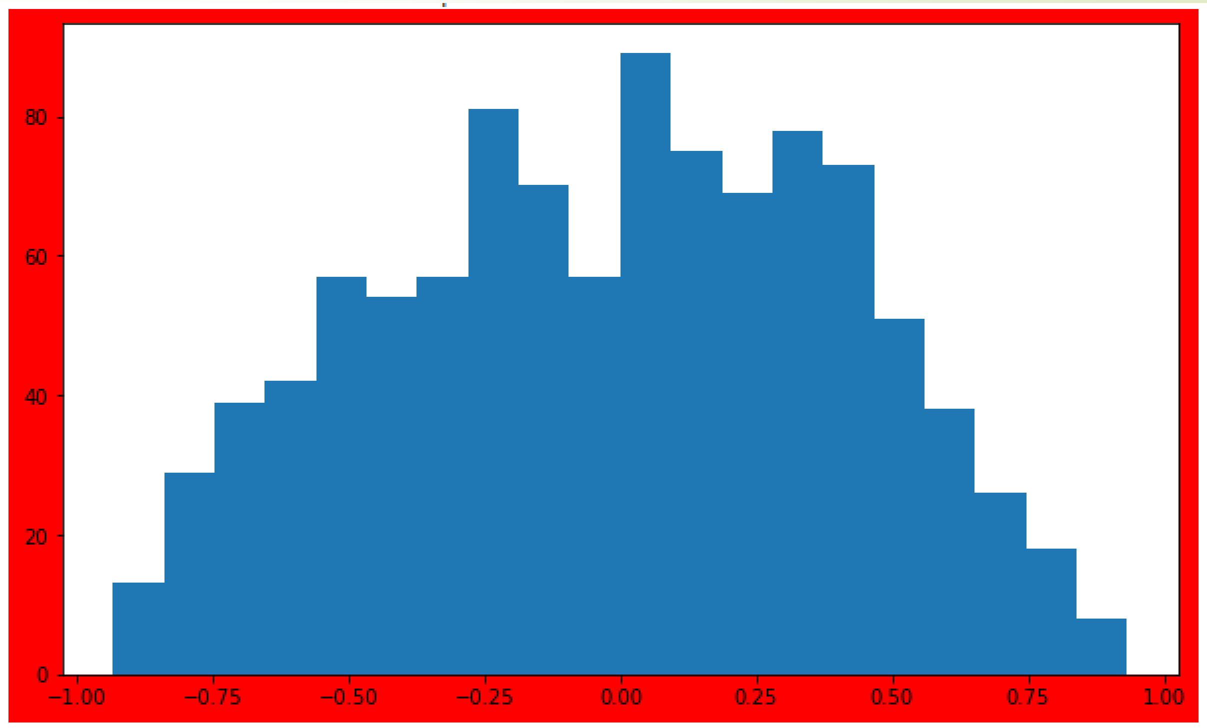


Glorot + tanh

glorot_uniform

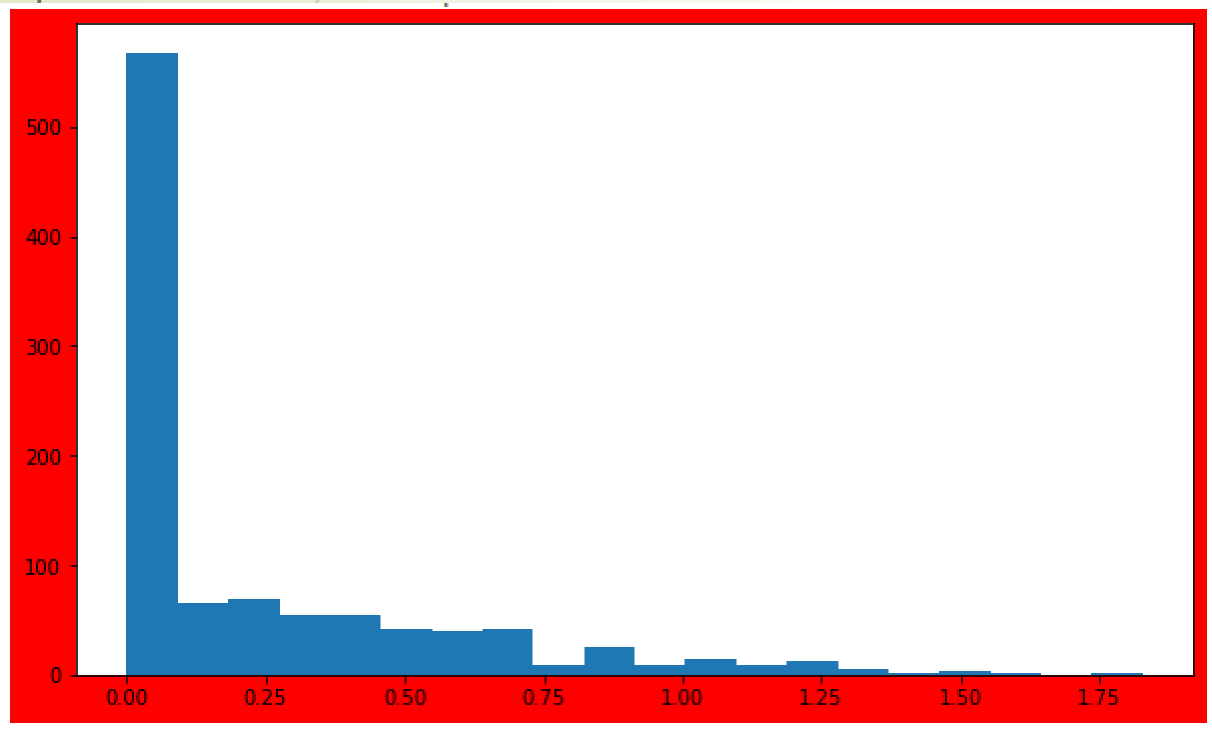


glorot_normal

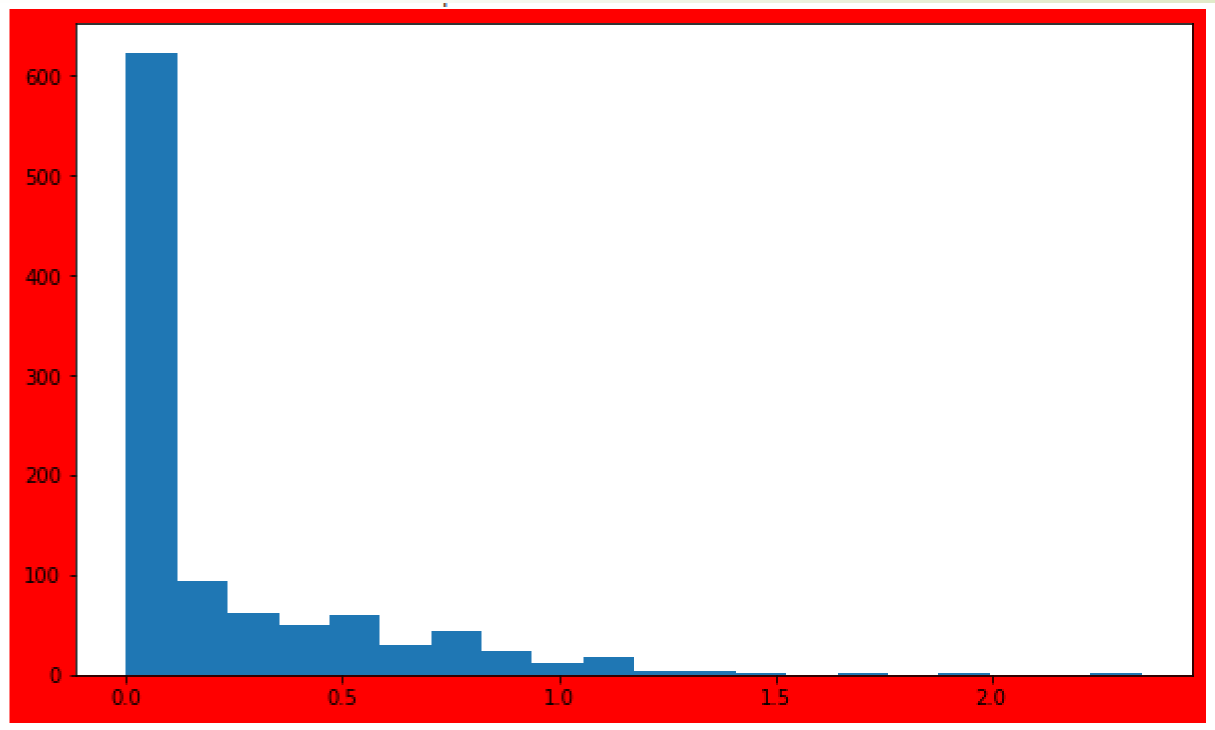


Glorot + relu

glorot_uniform



glorot_normal



Initialisation – summary

- It is not a simple technical matter
- Plays a crucial role in effective training
- If the signal gain is too large/small we can have exploding or vanishing gradients

- Proper initialisation is a key factor for deep models
- Glorot normal/uniform distributions have similar performance and can be used to prepare a very good starting point for any deep model

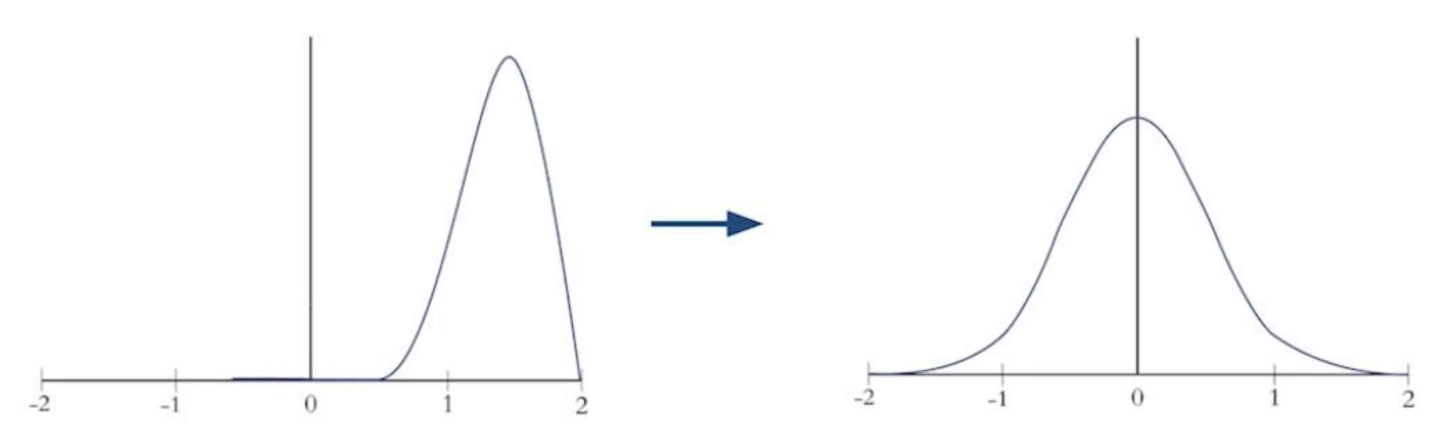
Unstable gradients

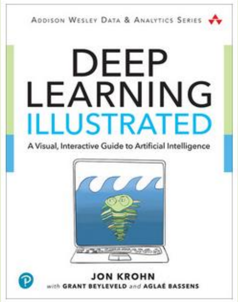
- Gradient descent is a primary technique applied to optimise the network response base on the loss function
- We adjust the parameters using their gradients with respect to the cost
- Intuition – **large gradient of a parameter = large contribution of this parameter to the cost**
- For back-prop there is a strong correlation between the layer position with respect to the network's output and the cost function
- The further away – the gradients of parameters tends to flat out (they are vanishing)
- Some deep models also exhibit exploding gradients (RNN)

Unstable gradients

- ❑ Both cases of extreme gradients are not desirable because they lead to neuron saturation, that impede the learning
- ❑ The studies of initialisation give us the hint regarding how the weights should behave
- ❑ It turns out that the most effective learning is possible when the parameter distribution is similar to the normal curve

- ❑ A new technique is thus introduced – **batch normalisation**





Batch „normalisation”

- Imagine, we start to get a **very distorted distribution** (called internal covariate shift) of outputs in a preceding layer – large values may drive the change of parameters in the next layer then
- To make the distribution **more balanced** we can transform it using **standardisation transform** – subtract the mean and divide by the variance – we remove extreme values
- To **protect against unwanted batch norm** we also add to each layer **two more trainable parameters γ and β**
- Using them, we can apply sort of reverse transform, in case the batch norm is not necessary and the **distorted weights actually are beneficial**
- NICE!

Batch „normalisation” - summary

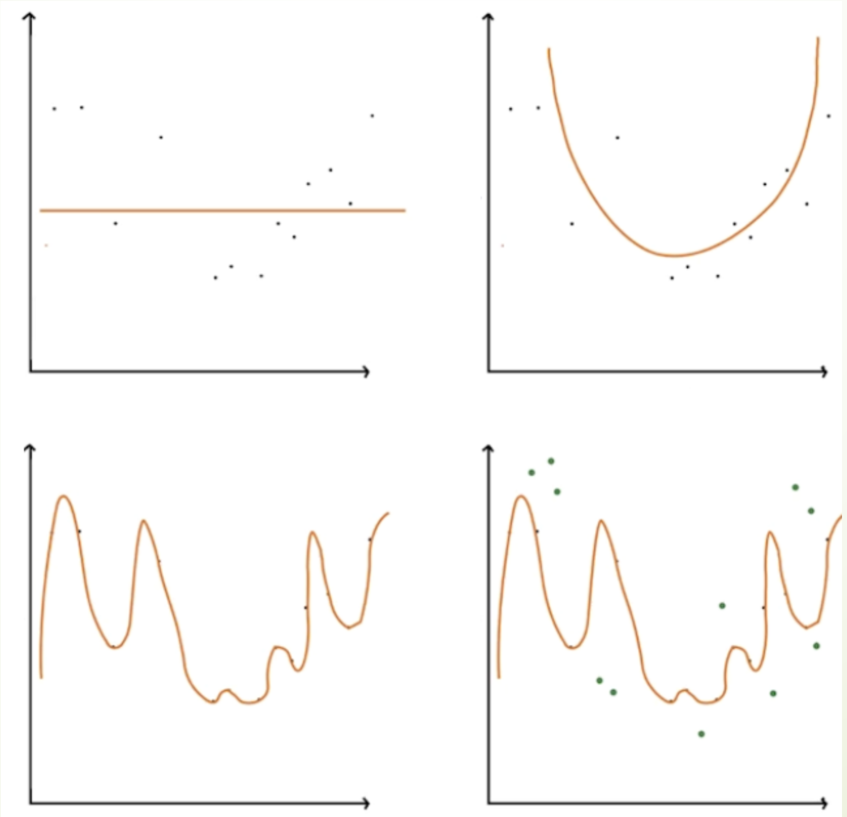
- **Batch norm may be a great tool to fine-tune deep networks**
- At the same time we add two more trainable parameters to the model to reverse this transform if necessary

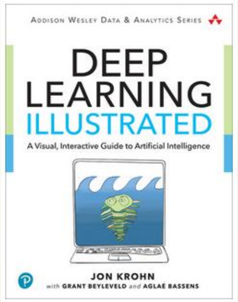
- Benefits
  - More **independent** learning from layer-to-layer
  - Can use **larger** learning rates
  - Batch norm can be viewed as „**noise adding operation**” what, in turn, may act as a **regularisation operation** and improve network generalisation properties (effect more pronounced for smaller batches!)

- $\boldsymbol{\gamma}$ **and** $\boldsymbol{\beta}$ are initialised to 1 and 0 respectively, the parameters will be adjusted by SGD

Over- and under-fitting

- One of the most disturbing behaviour of the cost function for train and test data set is opposite change
- It should be interpreter as model overfitting





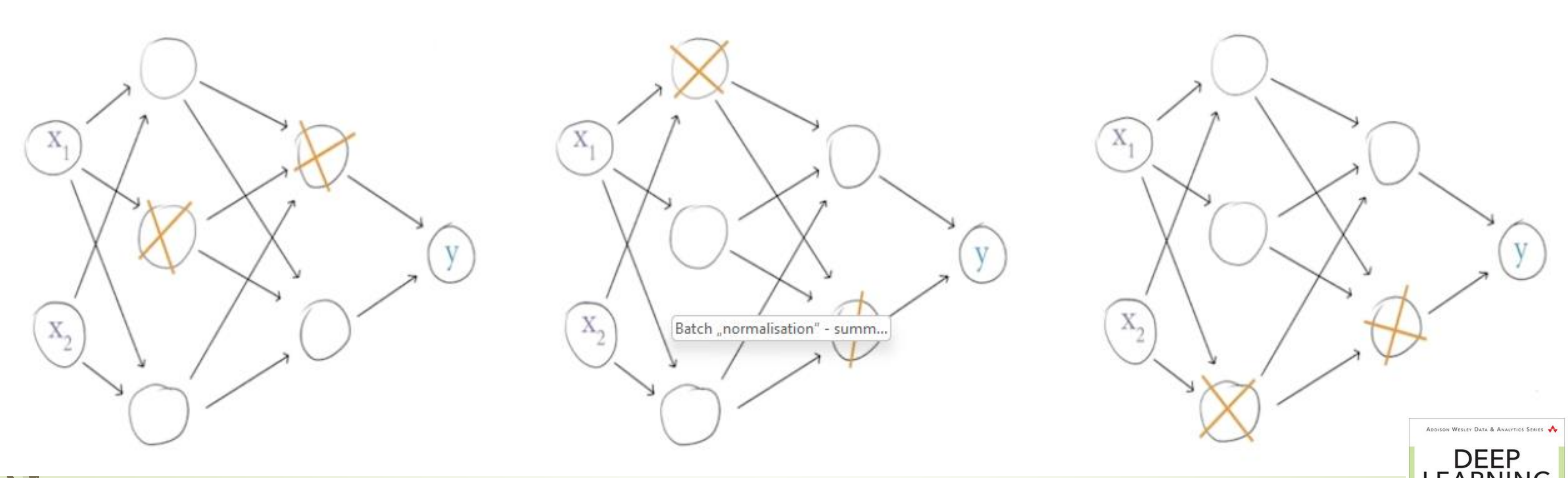
Over- and under-fitting

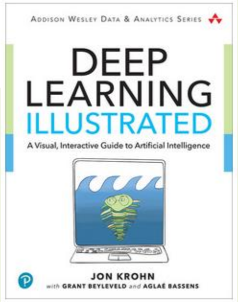
- There are three general techniques that can be broadly used for protecting against problematic training effects
  - Ridge/Lasso regularisation
  - Drop-out
  - Data augmentation (increase your data set!! The Keras will help you in that task!)

- The first set of techniques (called also L1/L2 regularisation) is used predominantly for simpler ANN models or different algorithms (i.e., decision trees)
- The remaining two are more specific to the deep models

Hinton’s drop-out

- ❑ Disruptive performance – AlexNET
- ❑ Ignore in each training cycle predefined portion of neurons





Hinton's drop-out

- Different strategy to batch normalisation
- Do not **control the value of neuron parameters** but rather prevent any neuron to become **too significant** in the network prediction
- Use only the **subset of neurons in forward-propagation** during training
- We do not want to create „strong" pathways in the network
- In other words – we protect against picking out something **very specific in the data** set that may influence the prediction
- This is going to help a lot in **generalisation** and in employing the **whole network** during its inference

- We target the differences between training and validation cost function

The end